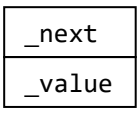
Практическое задание №15
(Односвязный линейный список)

Создать проект **ForwardList.Tests** (библиотеку создавать не нужно).

Необходимо реализовать **шаблонный класс** односвязного линейного списка **ForwardList**, параметризируемого типом хранимых в нём элементов.

Тип узла списка **ListNode** должен быть **шаблонным классом** (структурой) **ListNode**, параметризуемым типом хранимого в списке элемента. Можно тип ListNode определить внутри класса ForwardList. Структура ListNode должна иметь следующие поля:

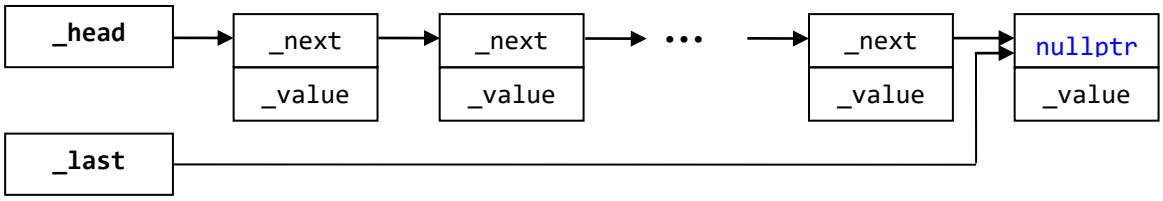
_next – указатель на следующий узел (типа ListNode *),
_value – хранимый в узле элемент списка (типа T)



и соответствующий конструктор с параметрами ListNode(ListNode* next, T value).
Запретить в ListNode создание тривиальных конструкторов копирования и перемещения, а также тривиальных операторов присваивания (с копированием и с перемещением).

В односвязном списке каждый узел хранит указатель на следующий узел списка. У последнего узла списка отсутствует следующий узел (указатель на него равен nullptr).

Класс **ForwardList** должен содержать указатель **_head** на первый узел списка (для пустого списка он равен nullptr). Для оптимизации вставки в конец списка целесообразно хранить и указатель **_last** на последний узел списка:



В классе **ForwardList** необходимо реализовать следующие методы:

1. Конструктор по умолчанию, создающий пустой список.
2. Конструктор копирования.
3. Конструктор переноса.
4. Метод void clear() noexcept, удаляющий все узлы списка.
5. Деструктор, освобождающий память, занятую узлами списка.
6. Метод ListNode * insert_after(const ListNode *where, const T& value), вставляющий новый узел с указанным значением value вслед за узлом where.
7. Метод void push_back(const T& value), добавляющий новый узел с указанным значением value в конец списка (путём соответствующего вызова метода insert_after).
8. Метод void push_front(const T& value), добавляющий новый узел с указанным значением value в начало списка (путём соответствующего вызова метода insert_after со значением where, равным nullptr).
9. Метод void remove(const T& value), удаляющий узел с указанным значением value из списка.

10. Метод ListNode * first(), возвращающий указатель на узел, соответствующий началу списка. Если список пуст, то должно выбрасываться исключение.
11. Метод ListNode * last(), возвращающий указатель на узел, соответствующий концу списка. Если список пуст, то должно выбрасываться исключение.
12. Метод T get(size_t position), возвращающий значение элемента, находящегося в заданной позиции.
13. Метод bool empty() const noexcept, возвращающий true, если в списке нет элементов, false – в противном случае.
14. Метод void reverse() noexcept, изменяющий порядок следования элементов списка на противоположный.
15. Метод size_t size() const, возвращающий количество элементов в списке.
16. Перегруженный в виде дружественной шаблонной функции оператор сравнения
    bool operator == (const ForwardList<T>& left,
    const ForwardList<T>& right).
17. Перегруженный в виде дружественной шаблонной функции оператор сравнения
    bool operator != (const ForwardList<T>& left,
    const ForwardList<T>& right).

Необходимо реализовать тесты для разработанных методов класса.

Для сдачи проекта использовать структуру в файловой системе:

- gxxxxx/15/Task15/ForwardList.h – файл с шаблонным классом;
- gxxxxx/15/Task5.Tests/*** – файлы с тестами пунктов задания (имена файлов давать в соответствии с вариантами).